DeepL Proに登録すると、より大きなサイズの文書ファイルを翻訳できます。
詳しくは、www.DeepL.com/pro をご覧ください。

パーマグループの効率的な表現

ドナルド・E・クヌース著
スタンフォード大学コンピューターサイエンス学部

マーシャル・ホールの思い出に捧ぐ

概要：このノートでは、与えられたパーマ群の強い生成子を計算するためのSimsのアルゴリズムの初等的なバージョンを、正しさの証明と適切な低レベルのデータ構造に関するいくつかの注意とともに紹介する。また、実行時間の上界と下界も求める。(Vaughan Prattの提案に従い、perm=permutationという慣例を採用し、おそらくそれによって将来の研究において何百万という音節を節約することができるだろう)。

1. パーマ・グループのデータ構造。パーマ」とは、本稿では、ある集合のそれ自身への1対1の写像である。αとβが、αがi '→j をとり、βがj '→k をとるようなパーマである場合、積$\alpha\beta$ はi '→k をとる。パーマαの逆数をα^- と書く。したがって、$\alpha = \gamma\beta^-$ であれば、$\alpha\beta = \gamma$ である。

$\Pi(k)$ を、すべての点$>k$ を固定する正の整数のすべてのパーマの集合とする：$1 \le j \le k$ のとき、$\sigma_{kj} = \emptyset$ またはσ_{kj} はk '→ j をとる$\Pi(k)$のパーマである。$\Sigma(k)$を∅でないすべてのパーマσ_{kj} の集合とする。σ_{kk} はidentity permであると仮定する。したがって、$\Sigma(k)$ は常に空でない。

各σ_i が$\Sigma(i)$の中にある$\sigma_1 . \ldots \sigma_k$ の形の積として書くことができるすべてのパーマの集合を $\Gamma(k)$ と書く。与えられたパーマ $\pi\in\Pi(k)$ が $\Gamma(k)$ のメンバーかどうかをテストする簡単な方法があります：もし$\pi_{kj} = \emptyset$ なら、$\pi \notin \Gamma(k)$がある。もし$k = 1$なら、$\pi\in\Gamma(k)$がある；
$\pi\sigma^- \in \Gamma(k - 1)$であれば$\pi\in\Gamma(k)$。

データ構造はまた、$\Gamma(k)$ の各要素が $T(k)$ の要素の積として書くことができるという不変特性を持つ集合 $T(k) \subseteq \Pi(k)$ を含む。言い換えれば、$\Gamma(k)$ は、これから

説明するアルゴリズムの過程を通して、すべての k に対して、$T(k)$ によって生成される群 ⟨T (k)⟩ の部分集合になります。($\Pi(k)$の全要素πは有限permsなので、ある$r > 0$について$\pi^{-} = \pi^{r}$ を持つ。したがって、乗算の下での閉包は反転の下での閉包を意味する)。

データ構造は、$\Gamma(k) \supseteq T(k)$であり、$\Gamma(k)$ が乗算の下で閉じている場合、すなわち $\Gamma(k) = \langle T(k) \rangle$で、$1 \leqq k \leqq n$の場合、次数$n$のアップツーデートであると言う。$S_n$ $\Sigma(k)$は$\Gamma(n)$の*横座標系*を形成し、「(k)の縦座標系を形成す$_{k=1}$ $T(k)$ は*強い生成子である*る。

のによってどのようなパーマが生成されるかを決定するのは簡単である。
与えられたパーマの集合T（n）。

2. データ構造の維持ここで、$T(k)$に新しいパーマが導入されたときにデータ構造を変換するために使用できる2つのアルゴリズムについて説明しよう。まずアルゴリズムを見てから、なぜそれが有効なのかを議論する。

本研究の一部は、全米科学財団（National Science Foundation）の助成金CCR-86- 10181および海軍研究局（Office of Naval Research）の契約N00014-87-K-0502の支援を受けている。

arXiv:math/9201304v1 [math.GR] 1991年1月1日

アルゴリズム$A_k(\pi)$。データ構造が次数 k の最新であり、$\pi \in \Pi(k)$ だが $\pi \notin \Gamma(k)$ であると仮定すると、この手続きは π を $T(k)$ に追加し、$\Gamma(k)$ が新しい $\langle T(k)\rangle$ と等しくなるようにデータ構造を最新に戻す。

ステップA1。πを集合T（k）に挿入する。

ステップ A2.$\sigma\tau$ が$\Gamma(k)$のメンバーであることが既に知られていないような、 $\sigma\in\Sigma(k)$および$\tau\in T(k)$のすべてについて、アルゴリズム$B_k(\sigma\tau)$を実行する。(アルゴリズムB_k 、$\Sigma(k)$のサイズが増加する可能性がある。$\Sigma(k)$に追加される新しいパーマσもこのステップに含まれなければならない。実装の詳細は後述のセクション3で説明する)。

アルゴリズム$B_k(\pi)$。データ構造が次数k - 1のアップトゥーデートであり、$\pi\in$が次数k - 1のアップトゥーデートであると仮定する。
$\langle T(k)\rangle$、この手続きはπが$\Gamma(k)$の中にあり、データ構造がk - 1の最新のままであることを保証する(kの値は常に1より大きい)。

ステップB1。πをk '→jとする。

ステップB2。$\sigma_{kj} = \varnothing$ の場合、$\sigma_{kj} \leftarrow \pi$ とし、アルゴリズムを終了する。

ステップ B3.$\pi\sigma^{-} \in \Gamma(k-1)$ の場合、アルゴリズムを終了する。($\Gamma(k-1)$のメンバーかどうかのこのテストには

は上記セクション1で説明したと

おりである)。ステップB4.アルゴリズム

$A_{k-1}(\pi\sigma^{-})$ を実行する。

アルゴリズム $A_k(\pi)$の終了時に$\Gamma(k)$が乗算で閉じていることを検証しなければならない。これは $k = 1$ のときに明らかなので、$k > 1$ と仮定してもよい。$\Gamma(k)$ の定義により $\alpha = \gamma\sigma$ と書くことができ、ここで $\gamma\in\Gamma(k-1)$ と $\sigma\in\Sigma(k)$; そして不変関係 $\Gamma(k) \subseteq \langle T(k)\rangle$↪Pe_27E9 により $\beta = \tau_1 \ldots \ldots \tau_r$ と書くことができ、ここで各 $\tau_i \in T(k)$ とする。ステップA2により、$\sigma\tau_1 \in \Gamma(k)$ であることがわかる。したがって、$\sigma\tau_1 = \gamma_1 \sigma_1$ は、ある$\gamma_1 \in$

$\Gamma(k-1)$とある$\sigma_1 \in \Sigma(k)$に対して成り立つ。同様に、$\sigma_1 \tau_2 = \gamma_2 \sigma_2$、などとなり、最終的に $\alpha\beta = \gamma_1 . \gamma_r \sigma_r$.これは、$\gamma \gamma_1 . \gamma_r$ は帰納法により $\Gamma(k-1)$ にある。

3. Low-level implementation hints. Let $s(k)$ be the cardinality of $\Sigma(k)$ and $t(k)$ the cardinality of $T(k)$. The algorithms of Section 2 can perhaps be implemented most efficiently in practice by keeping a linear list of the perms $\tau(k, 1) \ldots \tau k, t(k)$ of $T(k)$, for each k, together with an array of pointers to the representations of each σ_{kj} for $1 \leq j < k$, using a null pointer to represent the relation $\sigma_{kj} = \varnothing$. It is also convenient to have a linear list $j(k, 1) \ldots j\ k, s(k)$ of the indices of the non-$\varnothing$ perms σ_{kj}, where $j(k, 1) = k$. We will see below that the algorithm often completes its task without needing to make many of the sets $\Sigma(k)$ very large; thus most of the σ_{kj} are often $\varnothing$. Pointers can be used to avoid duplications between $T(k)$ and $T(k-1)$.

ステップA2のσとτのループを処理するには、再帰的な方法と反復的な方法の2つがある。再帰的な方法は、ステップA2を以下の操作で置き換える："現在の集合$\Sigma(k)$内のすべてのσについて、アルゴリズム$B_k(\sigma\pi)$を実行する。"そして、ステップB2も次のように変更される。"$\sigma_{kj} = \varnothing$ の場合、$\sigma_{kj} \leftarrow \pi$ と設定し、現在の集合$T(k)$ 内のすべてのτに対して$B_k(\pi\tau)$ を実行し、アルゴリズムを終了する。"

反復法は、ステップ A2 でどのペア (σ, τ) が既にテストされたかを記憶するために、追加のテーブルを保持する。このテーブルは、積$\sigma_{kj(k,i)}\ \tau(k, l)$が$1 \leqq l \leqq c(k, i)$で$\Gamma(k)$にあることが知られているような、各kと$1 \leqq i \leqq s(k)$のカウント$c(k, i)$から構成される。とき

step B2 increases the value of $s(k)$, the newly created count $c\ k, s(k)$ is set to zero. Step A2 is a loop of the form

```
i ← 1;
while i ≤ s(k) do
    begin while c(k, i) < t(k) do
        begin l ← c(k, i) + 1;
        B_k σ_kj(k,i) τ (k, l) ;
        c(k, i) ← l;
        end;
    i ← i +
    1; end;
```

B_k の呼び出しは$s(k)$を増加させるかもしれないが、$t(k')$と$c(k', i')$をkの値に対してのみ変化させることができる。kより小さい。

反復法は、再帰法とは異なる順序でテストを行うので、異なるトラバーサル・システムが得られるかもしれない。

It is convenient to represent each perm σ of $\Sigma(k)$ indirectly in an array q that gives inverse images, so that σ takes $q[i] \to i$ for $1 \le i \le k$. All other perms π can be represented directly in an array p, with π taking $i \to p[i]$ for $1 \le i \le k$. To compute the direct representation d of the product $\pi\sigma^{-}$, we can then simply set $d[$ 计 $q\ p[i]$ for $\le\ i \le k$. To compute the direct representation d of the product $\sigma\pi$, we set $d\ q[i] \leftarrow p[i]$ for $1 \le i \le k$. Thus, the elementary operations are fast.

4. 実行時間の上限。更新アルゴリズムの "内部ループ "はステップB3、メンバシップテストで発生する。$\pi \in \Gamma(k)$のメンバーシップのテストは、以下の乗算を含む。非同一パーマσのあるシーケンス $\sigma_{k\,j_{11}}^{-}, \cdots, \sigma_{k\,j_r}^{-}$ ここで、$k \ge k_1 > - - - > k_r > 0$; したがって、実行は時間は本質的に$k + k_1 + - - - + k_r$ に比例し、最悪の場合$O(k^2)$となる。

$B_k(\sigma\tau)$ の総実行回数は $s(k)t(k)$ であり、$s(k) \le k$ である。$t(k)$ の値は、$A_k(\pi)$を実行するたびに1ずつ増加する。これを実行するたびに、$\Gamma(k)$ を$\Pi(k)$のより大きな部分群に増加させるので、$t(k)$ は対称群$\Pi(k)$の部分群の最長の鎖の長さを超えることはできない。したがって、簡単な上界は$t(k) \le \theta(k!) = O(k \log\log k)$であり、ここで$\theta(N)$は$N$の素因数分解の多倍数の数である。Babai [1]は、Π(k)は、$k \ge 2$のとき、$2k - 3$を超える長さの部分群鎖を持たないことを示した。

アルゴリズムB_k ($\sigma\tau$)は$O(k^2)$回実行され、ステップB3の各出現には$O(k^2)$単位の時間がかかる。$1 \le k \le n$ について合計すると、*$\Pi(n)$ の m 個のパーマによって生成されるパーマ群に対する横座標系は、最大で* $O(n^5) + O(mn^2)$ *ステップで求めることが*できると結論づけることができる。($O(mn^2)$という用語は、アルゴリズムA_n (π)が適用される前に、各生成者π*に対して*行われるm*個の*メンバシップ・テストから来ている)。

$\Sigma(k)$または$T(k)$の各非同一パーマに必要な記憶容量は$O(k)$である。したがって、$\Pi(k)$のパーマに必要なメモリセルは最大で$O(k^2)$、*全体で*$O(n)$ *である。*[3]

5. 疎な例。このような手順で実際に計算しても、以下のような時間はほとんどかからない。

私たちの最悪のケースの推定が予測するものである。特定のケースを詳しく研究することで、真の効率についてより詳しく知ることができる。そこでまず、1つの非同一パーマ$\pi \in \Pi(n)$によって生成されるグループの場合を考えてみよう。

もちろん、$1 \leqq j < k \leqq n$の場合は$\sigma_{kj} = \varnothing$、$1 \leqq k \leqq n$の場合は$T(k) = \varnothing$で始める。データ構造は次数$n$のアップ・トゥ・デイトとなり、$A_n(\pi)$を実行できる。ここでπは $n \to a_1 \to \cdots \to a_{r-1} \to n.$ ならば、$A_n(\pi)$ は、$T(n) \leftarrow \{\pi\}$ および$\sigma_{na}^{j} \leftarrow \pi$ を $1 \leqq j < r$ とする。

は、$A_{n-1}(\pi^r)$ を呼び出す（π^r がアイデンティティ・パーマでない限り、その場合はアルゴリズムが終了する）。

例えば

$$\pi = [1, 2, 3, 4, 5, 6, 7, 14]\ [8, 9, 10, 13]\ [11, 12]$$

をサイクル形式とすると、アルゴリズムは、$1 \leqq j < 8$に対して$\sigma_{14,j} \leftarrow \pi^j$ を設定し、$T(14) = \{\pi\}$で終了する。

であり、他の$T(k)$はすべて空である。しかし、点12と点14のラベルを貼り替えると、共役のperm

$$\pi^{-} = [1, 2, 3, 4, 5, 6, 7, 12]\ [8, 9, 10, 13]\ [11, 14]\ ,$$

アルゴリズムは全く異なる動作をする：非自明なperms σ_{kj} と集合$T(\mathrm{k})$は次のようになる。

$\sigma_{14,11} = \pi^{-}$、$T(14) = \{\pi^{-}\}.,$　　$\sigma_{13,9} = \pi^{-2}$、$T(13) = \{\pi^{-2}\}.,$　　$\sigma_{12,4} = \pi^{-4}$；$T(12) = \{\pi^{-4}\}.$

アルゴリズムが終了すると、与えられたパーマρがそれぞれπまたはπ⁻のべき乗であるかどうかをテストできる横座標系が生成される。最初のケースでは、このメンバシップテストは、ρが$14 \to j$（$j < 8$）を取る場合、$\sigma_{14,j}$、最大でも1回の乗算を伴う。2番目のケースでは、テストは、例えば、$\rho = \pi^{-7}$ の場合、3回の乗算を伴う。

これらのパーマπとπ⁻は、次数のパーマの無限族の$h = 4$の特別な場合である。$n = 2^h - 2$で[illegible]り、長さ2^{h-1}、2^{h-2}、 ...、2 のサイクルを持つ。[1]一般に、πは～[2]

$\frac{1}{2}n$ *個の*スロットσ_{kj} を空でない状態にし、~ $\frac{1}{2}n^2$ の 初等機械ステップを実行した後に終了する。対応するperm π^{-} は、~ lg n *個の*スロットσ_{kj} だけが空でなくなり、~ $2n$ lg nステップの後に終了し、最悪の場合の実行時間が~ n lg n *である*メンバシップテストをもたらす。

6. 密な例。このアルゴ���ズムは、$\{\pi_2, \pi_3, \ldots, \pi\}$によって生成されるグループを見つけたいときに、より難しく働く必要がある。$_n$ここで、$\pi_k \in \Pi(k)$は$k' \rightarrow k - 1$をとり、生成子π_k *は*k*の*増加順に入力される。この場合、アルゴリズムが$T(k) = \{\pi_2, \ldots\}$で終了することを帰納法で検証するのは難しくない。$\pi_k\}$ であり、$1 \leqq j < k \leqq n$ で$\sigma_{kj} \neq \emptyset$ である。従って、アルゴリズムは全てのスロットσ_{kj} を埋めるので、*各*$\Gamma(k)$が完全な対称群$\Pi(k)$であることが暗黙のうちに推論される。さらに、ステップA2を実行するためにセクション3の再帰的方法が使用される場合、アルゴリズムは、ステップA2の実行に使用される。

で終了する。

$\sigma_{kj} = \pi_k \pi_{k-1} \ldots \pi_{j+1}$ 、　　とする。

σ_{kj} が定義された後、変更されたステップB2は、perms.

$\sigma_{kj}\ \pi_2$ 、　　$\sigma_{kj}\ \pi_3$ 、　.. .,　　$\sigma_{kj}\ \pi_k$

は現在の$\Gamma(k)$に属する。$B_k(\sigma_{kj}\,\pi_j)$は$\sigma_{k,j-1}$を定義させる。そして、$B_k(\sigma_{kj}\,\pi_j)$の再帰呼び出しが$B_k(\sigma_{kj})$に制御を返す頃には、σ_{ki}の値はすべての$i < k$に対して非∅になります。したがって、$l > j$に対する$\sigma_{kj}\,\pi_l$の残りのテストは成功します。

各π_kが単純な転置$[k, k-1]$であるこの構築の特別なケースを検証してみよう。$\Theta(n^3)$のメンバシップ・テスト$\sigma_{kj}\,\pi_i\,\Gamma(k)$にはどれだけの時間がかかるだろうか? 次のようになります。

$$\sigma_{kj} = [j, j+1, \ldots, k]\text{、}$$

となり、次のようになる。

$$\sigma_{kj}\,\pi_i = \begin{cases} \sigma_{i,i-1}\,\sigma_{kj}\text{、} & \text{if } 1 < i < j\text{;} \\ \sigma_{ki}\text{、} & i = j+1 \text{ の場合;} \\ \sigma_{i-1,i-2}\,\sigma_{kj}\text{、} & i > j+1 \text{ の場合。} \end{cases}$$

従って、各メンバーシップ・テストには、非同一性(non-identity perms)による乗算が最大でも2回含まれ、アルゴリズムの総実行時間は$\Theta(n^4)$である。

もう一つの興味深い特別なケースは、各π_kが環状perm $[k, k-1, \ldots, 1]$である場合に起こる。ここで、σ_{kj}は次のようになることがわかる。

$$\vec{x}' = \begin{cases} x-(k-j)\text{、} & x > k-j\text{;} \\ x+1-x\text{、} & x \le k-j \text{ の場合。} \end{cases}$$

その結果、我々は

$$\sigma_{kj} = \begin{cases} \sigma_{k-j,1}\,\sigma_{k-j+1,1}\,\sigma_{k-j+i,k-j+i-1}\,\sigma_{kj}\text{、} & i < j \\ \sigma_{k-i,1}\,\sigma_{k-j,1}\,\sigma_{k-j+1,k-i+1}\,\sigma_{k,j-1}\text{、} & \text{もし } 1 < j < i < k \text{ ならば;} \\ \sigma_{k-i,1}\,\sigma_{k-2,i-2}\,\sigma_{k-1,k-i+1}\,\sigma_{ki}\text{、} & \text{の場合;} \\ \sigma_{k2}\text{、} & \text{が成り立つ;} \\ \sigma_{k-1,1}\text{、} & \text{となる;} \\ \sigma_{k-j,1}\,\sigma_{k-j+1,1}\,\sigma_{k,j-1}\text{、} & \text{となる;} \\ & \text{もし } 1 < j < i = k \text{ ならば} \end{cases}$$

そのため、メンバシップ・テストにはそれぞれ最大4回の乗算が必要であり、総実行時間は$\Theta(n^4)$となる。

これらの特殊なケースのいずれにおいても、ステップA2の反復的な実装は、同じperms σ_{kj}を定義することがわかる。したがって、実行時間は、これまで議論してきたい

ずれの実装においても$\Theta(n^4)$となる。

もう一つの特別なケース、すなわち、σ_n =[1、2、 ...、n]とτ_n =[n-1、n]の2つだけの生成子がある場合のアルゴリズムを分析するのは興味深い。再帰的実装が使われると仮定する。まず、アルゴリズム$A_n(\sigma_n)$ は、$T(n) = \{\sigma_n\}$ を設定し、アルゴリズム$B_n(\sigma_n)$ を実行する。アルゴリズム $B_n(\sigma_n)$ は $\sigma_{n1} \leftarrow \sigma$ をセットする。 $_n$
を設定し、$B_n(\sigma^2{}_n)$を実行し、$\sigma_{n2} \leftarrow \sigma^2{}_n$を設定し、$B_n(\sigma^3{}_n)$などを実行する。従って、
σ_{nj} は次のようにσ^j となる。
次に、アルゴリズム$A_n(\tau_n)$ は、τ_n を$T(n)$ に追加し、$B_n(\tau_n)$, $B_n(\sigma_n \tau_n)$, ... を実行する。$B_n(\sigma^{n-1} \tau_n)$ を実行する。これらのサブルーチンの最初のもの、$B_n(\tau_n)$ は、アルゴリズム$A_{n-1}(\tau_n \sigma_n)$ を実行し、これは$A_{n-1}(\sigma_{n-1})$ である。番目のサブルーチン、$B_n(\sigma_n \tau_n)$ は、$A_{n-1}(\sigma_n \tau_n \sigma^{-1})$を実行し、これは$A_{n-1}(\tau_{n-1})$である。したがって、すべてのjと$k$に対して、$\sigma_{kj} = \sigma^j$ であることを示すために、nに対する帰納法を使用することができる。各メンバーシップ・テストは、最大でも3つの自明でない乗算を必要とすることを検証するのは簡単である。従って、$\Gamma(n)$は完全な対称群$\Pi(n)$であるが、この特別な場合の総実行時間は$\Theta(n^3)$だけである。

7. ランダムな例。セクション6の構成条件は、各 perm π_k に対して(k-1)！の可能性を許す。

1！ 2!. . .(*n* - 1)！ {π_2 , π_3 , . .π_n }は同じ確率である。直感的には、平均実行時間はΘ(n^5)であると考えるのが妥当である。これは、少なくともステップA2の再帰的実装を使用した場合、実際に正しいことがわかる。

前と同様に、実行時間は、$\Gamma(k)$における$\sigma_{kj}\ \pi_i$ のメンバシップに対するΘ(n^3)個の成功したテストに支配される。ここで、$k > j \geq 1$ かつ $k \geq i > 1$ かつ $i \neq j$ である。総実行時間は $O(n^5)$であることが分かっているので、平均値が Ω(n^5)であることだけを示せばよい。

$\sigma_{kj}\ \pi_i$ のメンバシップ・テストは、次の乗算を実行する。

$$\sigma_{kj}\ \pi_i\ \sigma^{-}_{kj_k}\ \sigma^{-}_{\mathrm{k\text{-}1},j_{k-1}}\ \ldots\ \sigma^{-}_{2j_2}$$

そして、$j_l \neq l$ となるような各乗算のコストはl である。$j > i$ であるため、常に$j_k = j$ となる。k、j、 i、lの値を固定し、$k > j > i > 1$、$k > l > i$とし、$j_l = l$となる確率の上限を決定してみよう。以下の分析は、任意の（必ずしもランダムではない）パーマのシーケンスπ_l , .π_2 , π_k , .π_{l+1} はランダムに変化する。

与えられたパーマπ_i によって固定される点≦i の数を i - r とする。$_i$
はi '→ i - 1、したがってr≧2である。

最初の目標は、$j_{k-1} = \mathrm{k\text{-}1}, j_{k-2} = \mathrm{k\text{-}2}, \ldots j_l = l.$
これは、$\sigma_{kj}\ \pi_i\ \sigma^{-}$ Π(*l* - 1)の場合に成り立つ。ステップA2の再帰的実装では、次のようになる。

$$\sigma_{kj}\ \pi_i\ \sigma^{-}_{kj} = \pi_k\ \pi_{k-1} \ldots \pi_{j+1}\ \pi_i\ \pi^{-}_{\mathrm{j+1}} \ldots \pi^{-}_{\mathrm{k\text{-}1}\ k}\ \pi^{-}_{k} = \pi_k\ \rho\pi^{-}_{k}、$$

ここでρはπ_i と同じサイクル構造を持つ$\Pi(k - 2)$のpermであり、したがってρはk - 2 - r点≦ k - 2を正確に固定する。π_k がその(*k* - 1)！可能な値を通るとき、$\pi_k\ \rho\pi^{-}$ に何が起こるかを考える：例えば、r = 7でρ = [1 2 7][3 6][4 9]の場合、(*k* - 1)！ $\pi_k\ \rho\pi^{-}$ パームは、[$a_1\ a_2\ a_7$][$a_3\ a_6$][$a_4\ a_9$] as a_1 ... a_{k-1} 、Π(*k* - 1)の全パームのイメージを通る。したがって
$\sigma_{kj}\ \pi_i\ \sigma^{-}_{kj}$ Π(*l* - 1)は以下の通りである。

$$\begin{matrix} l\text{ - }1 \\ r \end{matrix}\ \begin{matrix} k\text{ - }1 \\ r \end{matrix} = (\frac{l\text{ - }1)(l\text{ - }2)\ldots(l\text{ - }r)}{(k\text{ - }1)(k\text{ - }2)\ldots(k\text{ - }r)}$$

ここで、j_{k-1} = k - 1, ... , j = q +1, j < q, j = l となる確率を計算してみよう。

$j_{k-1q+1} = q+1, j_q < q, j_l = l, k > q > l$ の範囲に添え字qがあると仮定する。π_{q+1}，π_{q-1}，., π_2 have been assigned some fixed values, while π_k and π_q run independently through all of their (k − 1)!(q - 1)!
の可能性がある。このような状況下で、$\sigma_{kj}\,\pi_i\,\sigma^{-}\,\sigma^{-}$ が一様に分布することを証明する。
$kj\ qjq$

$\Pi(q\text{-}1)$以上で
ある。

$\alpha \in \Pi(q)$が$q' \to p$をとり、π_i と同じサイクル構造を持つとする。 また、βを$\Pi(q - 1)$の要素とする。すると、次のようなパーマπ_q がちょうど1つ存在する。

$\alpha\sigma_{qp}$を作る$^{-} = \beta$、すなわち

$$\pi_q = \beta^{-}\,\alpha\pi^{-}_{p+1} .. \pi^{-}_{q-1} \quad .$$

(このパーマは$q' \to q-1$をとり、すべての点$>q$を固定する。

π_qと呼ぶ)。さらに、π_qがこの値を持つとき、$\sigma_{kj}\,\pi_i\,\sigma^- = \alpha$となるパーマ$\pi_k$の数は

前のケースで観察したように、αとは無関係である。したがって、($j_q = p$
そして$\sigma_{kj}\,\pi_i = \beta$）は$\beta$に依存せず、$p$にも依存しない。

$\sigma_{kj}\,\pi_i\,\sigma^-\,\sigma^-$の一様分布は、（$j_k, j_{q-1} \neq k-1, \ldots, j_{q+1} = q+1$、

$j_{q-1} = k-1, \ldots, j_{q+1} = q+1, \text{and } j_q < q$）である確率の$1/l$倍の確率で、（$j_q < q, \text{and } j_l = l$
）である。j_2は一様に分布しているからである。そして、この確率が

$$\frac{1}{l}\,\frac{(q-1)\ldots(q-r+1)}{(k-1)(k-2)\ldots(k-r)} - \frac{(q-1)(q-2)\ldots(q-r)}{(k-1)(k-2)\ldots(k-r)} = \frac{r\,(q-1)\ldots(q-r+1)}{l\,(k-1)\ldots}.$$

最後に、k、j、i、lが上記のように与えられ、π_iが$r-1$個の固定点を持つとき、$j_l = l$となる確率を計算することができる：それは次のようになる。

$$\frac{1}{(k-1)\ldots}\,(l-1)\ldots(l-r) + \frac{r}{l}\sum_{l<q<k}(q-1)\ldots$$

$$= \frac{1}{l} + \frac{(l-1)\ldots(l-r)(l-r-1)}{(k-1)\ldots(k-r)(l-r)}$$

$$< \frac{1}{l} + \frac{(l-1)\ldots(l-r)}{(k-1)\ldots}.$$

$r \geqq 2$なので、次のような上限が得られる。

$$\Pr(j_l = l) < \frac{1}{l} + \frac{(l-1)(l-2)}{(k-1)(k-2)} < \frac{1}{l} + \frac{l^2}{k^2}.$$

これは総乗算時間に対する望ましい下界$\Omega(n^5)$を意味する。例えば、$1 < i \leq 1^n < l \leq 1^n < j \leq 3^n < k \leq n$の$\Omega(n^4)$の値$(k, j, i, l)$の和を求めることができる。

の場合、乗算は少なくとも$1 - (1/l + l^2/k^2) > 1/2$の確率で$\Omega(n)$ステップを必要とする。
$n \geq 72$.

平均実行時間は$\Omega(n^5)$であるため、すべてのnについて、アルゴリズムに$\Omega(n_2)$の演算を行わせるpermsのシーケンス$\pi_1, \ldots, \pi_n$が存在しなければならない。$^5\pi_n$が存在するはずである。しかし、そのようなpermsを明示的な構成によって定義することは難しいようである。また、完全にランダムな場合であっても、再帰的な実装の代わりにステッ

プA2の反復的な実装を採用した場合に、$\Omega(n^5)$の境界を証明する明白な方法はない。

8. より意味のある上限。上記の例から、アルゴリズムAやBを単に「$O(n^5 + mn^2)$のワーストケース実行時間で$\Pi(n)$のm個のパームを処理する」と言うだけで特徴づけるのは誤解を招くことがわかる。$\Omega(n^5)$の動作が実際に起こりうることがわかったので、ある意味ではこの見積もりはシャープである。

もう1つのパラメータを導入することで、セクション4の推定を改善することができる。生成される群$\Gamma(n)$の次数をgとする。すると次の結果が得られる：

Theorem. *A transversal system for a perm group of order g generated by m perms of Π(n) can be found in at most* $O\ n^2(\log g)^3/\log n\ + O\ n^2(\log g)^2\ + O(mn \log g)$ *steps, using at most* $O(n^2 \log g/\log n) + O\ n(\log g)^2$ *memory cells.*

証明。$s(k)$と$t(k)$を前述のように定義する。すると$g = \prod_{k=1}^{n} s(k)$であり、メンバシップ数テストは $m + \sum_{k=1}^{n} s(k)t(k) - s(k) + 1$ である。各メンバーシップ・テストには最大$O(\log g)$の乗算が必要である。

なぜなら、$s(k) > 1$を持つ添字kの数は、gの素因数の総数である$\theta(g)$を超えることができないからである。

Moreover, each $t(k)$ is at most $\theta(g) = O(\log g)$, as we have argued before. Therefore we can

が、$\sum_{k=1}^{n} s(k) - 1 = O(n \log g/\log n)$.

nとsが与えられたとき、積$\prod_{k=1}^{n} s_k$を最小化することを試みる。以下の条件に従う。

$$s = \sum_{k=1} (s_k - 1) \qquad \text{であり } 1 \le s_k \le k \text{ 。}$$

$s_{k-1} > s_k$ ならば、条件に違反することなく $s_{k-1} \leftrightarrow s_k$ を入れ替えることができる。したがって、$s_1 \leqq s_2 \leqq \text{- -} \leqq s_n$ と仮定することができる。 さらに、$1 < s_{k-1} \leqq s_k < k$ ならば、$(s_{k-1}, s_k) \leftarrow (s_{k-1} - 1, s_k + 1)$ とすることで積を小さくすることができる。したがって、積が最小になるのは、できるだけ大きな k に対して $s_k = k$ となるときである：

$$s_n = n、\ \ s_{n-1} = n - 1、\ \ \ldots,\ \ \ s_{q+1} = q + 1、\ \ s_q = r、\ \ s_{q-1} = \text{- - -} = s_1 = 1.$$

ここで、q と r は次のようなユニークな整数である。

$$n - s - 1 = q - r \qquad 2 \qquad \text{であり } 1 \le r < q \le n \text{ である。}$$

(ここでは $0 \le s < n$ と仮定する。) 最小積は次のようになる。

$$P(n, s) = r\frac{n!}{!} \cdot q$$

アルゴリズムにおける実際の積は $g \ge P\ n, \sum_{k=1}^{n} s(k) - 1$ である。

if we can show that

$$s = O\ n \frac{\log P(n, s)}{\text{対数 } n}.$$

しかし、これは難しいことではない。$s \ge \frac{1}{4}n^2$ とすれば、$q \le n/\sqrt{2}$ となり、$\log P(n, s)$

$= \Theta(n \log n)$となる。

の結果が成り立つ。もう一方の極端な例として、$0 \leqq s < n$の場合、$P(n, s) = s + 1$となり、ここでも結果はトリビアルである。そうでない場合は、$n - q \geq \lfloor s/n \rfloor$であることに注意しよう。

$$P(n, s) \geqq \frac{n!}{q!} > q^{\lfloor s/n \rfloor} > \left(\frac{n}{2}\right)^{s/n-1};$$

とすると、$(s/n) \log n = O \log P(n, s)$の関係が直ちに成り立つ。

横方向のパーマσ_{kj}を保存するのに必要な容量は$\Sigma^{n}_{k=1} k\, s(k) - 1 = O(n^2 \log g/\log n)$。

強力なジェネレーターを保存するのに必要なスペースは、k個のジェネレーターを合計した$\Sigma\, k\, t(k)$になる。

$s(k) = 1$ならば、$T(k) = T(k - 1)$となる。この和は$O(\log g)$の項を持ち、それぞれの項は$O(n \log g)$である。これで定理の証明は完了である。

この証明の検査から、実行時間は実際には主張されているよりもわずかに小さい推定値、すなわち

$$O\ n^2 l_n(g)^2 \log_n g\ + O\ n^2 l_n(g)^2\ + O\ mnl_n(g)\ , \qquad \text{where } l_n(g) = \min\ n, \theta(g)\ .$$

空間境界は、同様に$O(n^2 \log n\, g) + O\, nl_n\,(g)^2$ である。そして、上記のセクション5と6 の 例は、この改善された境界でさえ過度に悲観的かもしれないことを示している。

強力なジェネレータが占有するストレージは、通常、トラバーサル・システムのパームに必要なストレージよりも小さいが、大きくなることもある。例えば、nが偶数で生成子がそれぞれ

$$[n - 1, n]$$
$$[n - 3, n - 2]\ [n - 1, n].$$
$$.$$
$$[1, 2]\ ..\ .[n - 3, n - 2]\ [n - 1, n]\ .$$

とすると、$g = 2^{n/2}$ となり、$n\ l_n\,(g)^2$ 項が支配的となる。

$l_n\,(g)$ と $\log n\, g$の値は、計算上興味のあるパーマ群では、しばしばnよりかなり小さい。例えば、Hall-Janko群J_2 は、$g = 2^7 - 3^3 - 5^2 - 7$、$n = 100$である（[6]を参照）；ここで$\theta(g) = 13$であり、$\log_n g \approx 2.9$である。次数$g = 2^{15} - 3^6 - 5 - 7 - 11$を持つユニタリー群$U_6(2)$は、ケイリーライブラリーの$n = 672$点上のパーマ群として表現される（[10]を参照）；この場合、$l_n\,(g) = 24$、$\log_n g \approx 3.5$。代表的な大きな例として、Conwayの完全群-0がある、に対して、$g = 2^{22} - 3^9 - 5^4 - 7^2 - 11 - 13 - 23$、$n = 196560$、$\log_n g \approx 3.6$;そしてFischerの単純群である。
$F_{24} = 2^{21} - 3^{16} - 5^2 - 7^3 - 11 - 13 - 17 - 23 - 29$, $n = 306936$, $\log_n g \approx 4.4$. (参照[3])。

9. 歴史的発言と謝辞。上記のアルゴリズムは、1967年にSimsによってスケッチされた基本的な手順[8]の変形であり、数年後、彼はより大きなアルゴリズム[9]の一部としてより完全に説明した。9]の方法と本手法の主な違いは、Simsが基本的に$T(1) \subseteq T(2) \subseteq \cdots \subseteq T(n)$という条件を満たす強生成子の集合を扱ったことである。彼の例では、[1, 2, 4, 5, 7, 3, 6]と[2, 4] [3, 5]で生成される群では54個の積$\sigma\tau$を検証する必要があったが、本

方式では40個の積を検証するだけでよい。一方、$\Sigma(k)$を生成器の単語として表現する彼の方法は、記憶空間の使用においてかなり経済的であり、当時、空間は極めて重要な資源であった。さらに、強力なジェネレータを維持する彼の方法は、彼のシステムの他のルーチンとうまく調和していたため、彼が本論文の方法を改善とみなしたかどうかは定かではない。

ワーストケースの実行時間に関する多項式境界は、このオリジナルな研究からは明らかではな かった。Furst, Hopcroft, and Luksは1980年[5]に、$O(n^6)$ステップで横断系と強生成器の集合を見つけられることを示した。(彼らの方法では、横座標系と強生成系は

は同一であった)。その1年後、著者は『*The Art of Computer Programming*』第4巻の執筆準備 中、またPersi Diaconis [4]の研究プロジェクトに取り組んでいた学部生Eric W. Hamiltonに助言を与えながら、現在のアルゴリズムを開発した。著者が1981年11月6日にオーバーウォルファッハで開催されたカンファレンスで非公式に議論した後、現在の手法がより広く知られるようになった。特にクレメント・ラムをはじめとする何人かの人々が、そのときに配布された大まかなメモの明確化を提案した。最終的にババイ教授は、第4巻が完成するまで待たずに、1981年のノートを今出版することを提案してくれた。これらのノートは、本論文のセクション1-4に若干の改良を加えて再録されている。著者は、査読者と馬場井教授とルクス教授に感謝している。セクション5-8の追加資料を促すような鋭い指摘をいくつかいただいたBabai教授とLuks教授に感謝する。

この間、改良された方法が発見され、特にJerrum [7]は、ワーストケースのストレージ必要量を n^2 のオーダーまで削減した。Babai, Luks, and Seress [2]は、ワーストケースの実行時間が$O(n^{4+o})$で済む、より複雑な手順を開発した。

著者のオーバーウォルファッハ・ノートで実験的に導入された「パーマ」という言葉は、改宗者を獲得していないようだ。(実際、プラット自身、著者との会話でかつてこの提案をしたことを忘れている)。しかし、通常のπ^{-1} の代わりにπ^{-} という表記法を逆数に使うという提案には大きなメリットがあり、著者は将来的にこの表記法が広く採用されることを期待している。短い表記法は黒板に書きやすく、キーボードでタイプしやすい。さらに、α^{1} が冗長であるように、より長い表記α^{-1} も 冗長である。実際、α^{-1} はα^{-} の1乗を表している！このように、2つの表記法の間に矛盾はなく、段階的な切り替えが可能である。

参考文献

[1] László Babai, "On length of subgroup chain in symmetric group," *Communications in Algebra* 14 (1986), 1729-1736.

[2] László Babai, Eugene M. Luks, and Ákos Seress, “Fast management of permutation groups,” *29th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, 1988), 272–282.

[3] J. H. Conway, "Three lectures on exceptional groups," in M. B. Powell and G. Higman, edings, *Finite Simple Groups*, Proceedings of the Oxford Instructional Conference on Finite Simple Groups, 1969 (London: Academic Press, 1971), 215-247.

[4] Persi Diaconis, R. L. Graham, and William M. Kantor, "The mathematics of perfect shuffles,".
Advances in Applied Mathematics 4 (1983), 175-196.

[5] Merrick Furst, John Hopcroft, and Eugene Luks, "Polynomial-time algorithms for permuta- tion groups," *21st Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, 1980), 36-41.

[6] マーシャル・ホール・ジュニア、デビッド・ウェールズ「次数604,800の単純群」 *『代数学雑誌*
9 (1968), 417-450.

[7] マーク・ジェラム，「並べ替え群のコンパクトな表現」，*Journal of Algorithms* 7 (1986), 60-78.

[8] Charles C. Sims, "Computational methods in study of permutation groups," John Leech, ed, *Computational Problems in Abstract Algebra*, Proceedings of a conference held at Oxford University in 1967 (Oxford: Pergamon, 1970), 169-183.

[9] Charles C. Sims, "Computation with permutation groups," in S. R. Petrick, ed., *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, Los Angeles, California (New York: ACM, 1971), 23-28.

[10] D.E. Taylor, "Pairs of generators for matrix groups," *The Cayley Bulletin* 3 (Department of Pure Mathematics, University of Sydney, 1987).